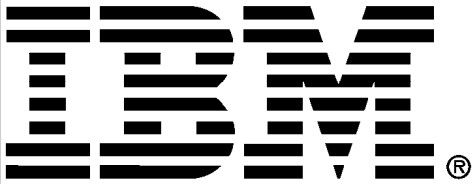
# The Snow Report:
## A GPFS Performance & Bottleneck Analysis

UCRL-VG-141522

**snow.llnl.gov**
**(bldg. 451)**

**William Loewe**
**Summer, 2000**
**Lawrence Livermore National Laboratory**



# Report Outline


- Background:
  - Snow Architecture in conjunction with GPFS 1.3
  - RAID Access Patterns
- File Creation/Removal Rates using NFS, JFS, GPFS 1.2 & GPFS 1.3:
  - Small Files
  - Large Files
  - Directories
- IOR_POSIX benchmark tests:

  Segmented Access, Varying:
  - Transfer Size
  - Node Number
  - File Size
  - Client-Node Ratio
  - Transfer Size w/Multiple Client
  - Random Size Transfer

  Strided Access, Varying:
  - Block Size
- Concluding Summary

# Snow Hardware Specifications


- IBM RS/6000 SP System
- 4 frames
- 4 Nighthawk-1 nodes per frame
- 8 222MHz 64-bit IBM Power3 CPUs per node

  (16 Nodes & 128 CPUs total)
- 4 GBytes RAM per node

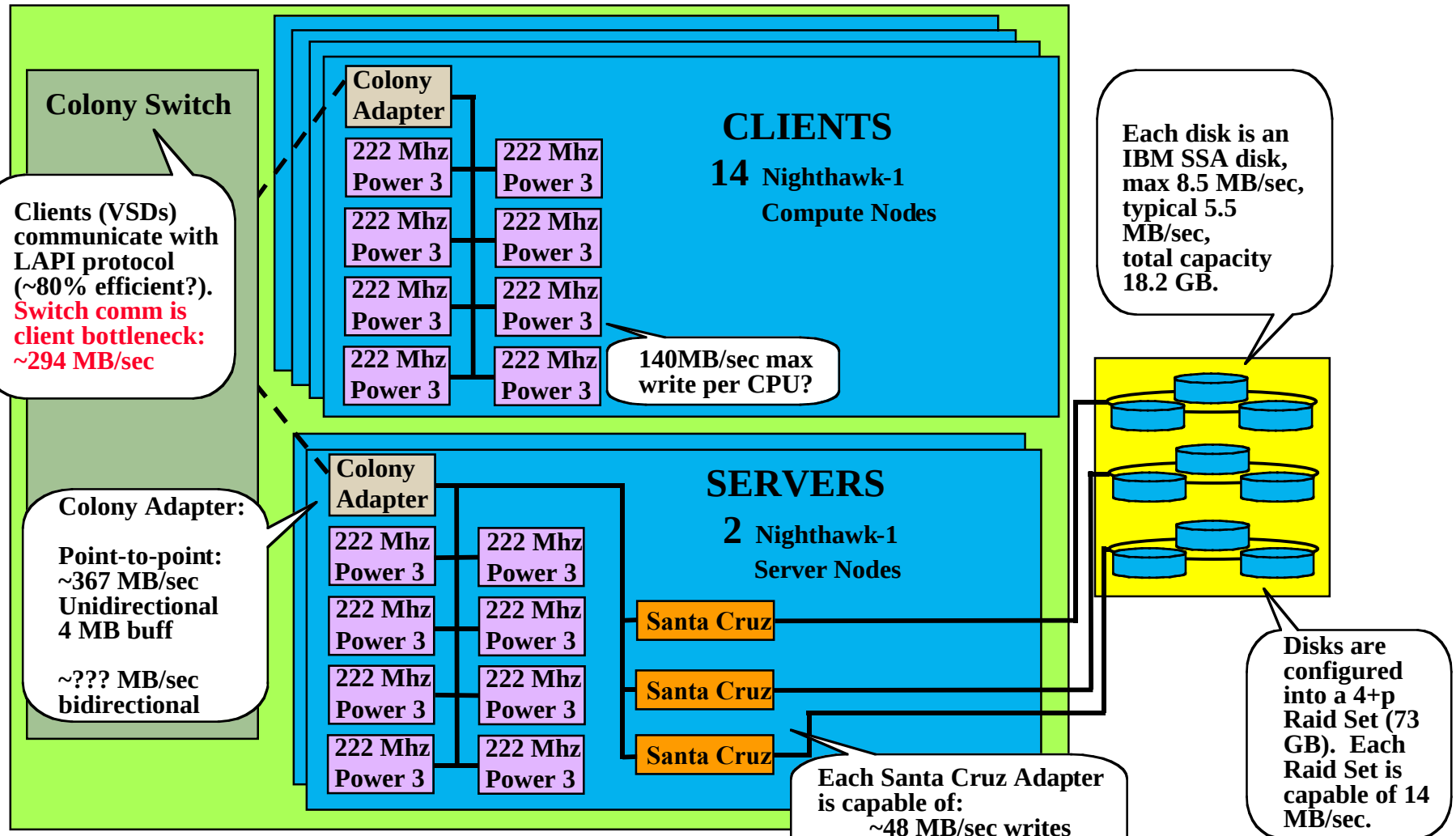
  (64 GBytes total memory)
- Peak computing capability of

  ~114 GFLOPS

- 2 I/O nodes (may also be used as compute nodes -- not dedicated)
- 3 SSA adapters per I/O node
- 3 73-GByte RAID sets per SSA adapter (1.3 TBytes total disk space)
- 5 Disks on each RAID (4 + P)

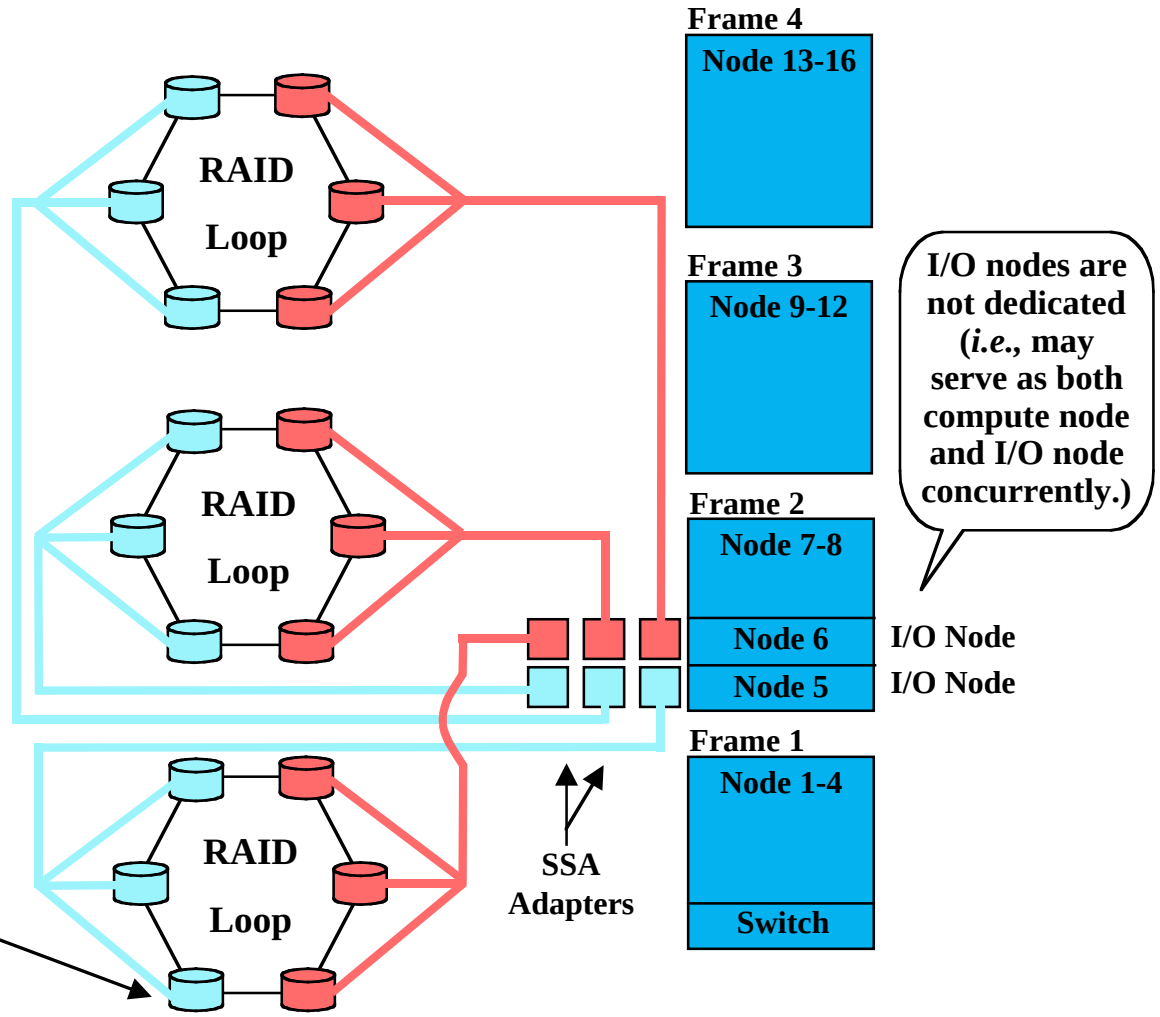
  – parity information is distributed among drives (i.e., not a dedicated parity drive)
- ~250MB/sec maximum transfer (14MB/sec per RAID)

# Analysis of NH-1 Nodes w/ Santa Cruz adapters



# RAID Loops

- RAID loops are shared between I/O nodes (5 and 6) so that if a single I/O node fails, another I/O node may cover all eighteen RAID sets.

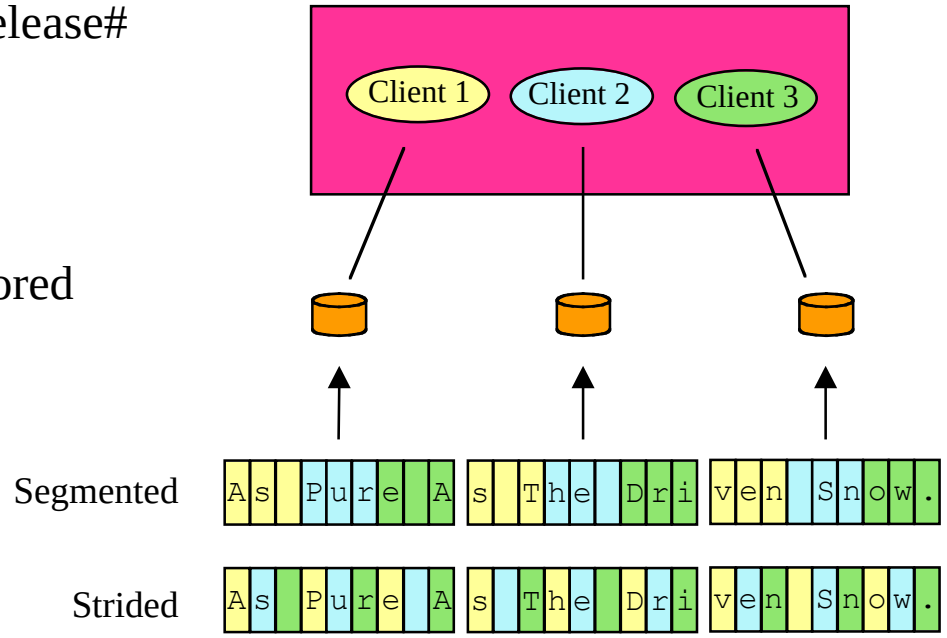


# GPFS version 1.3

- IBM General Parallel File System for AIX (GPFS) version 1.3
- Mohonk PSSP 3.2 GA PTF 1 SubRelease# DV (GPFS)

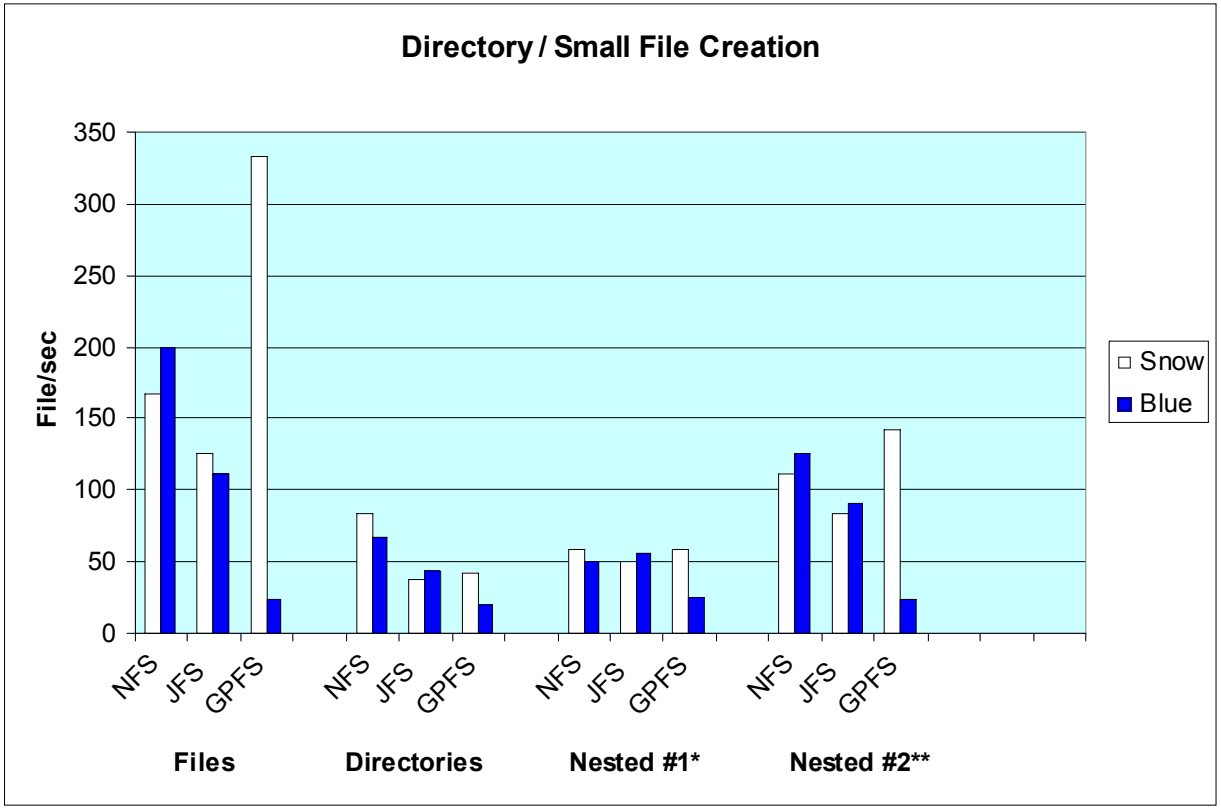
Primary goal is to allow a single file accessed by multiple clients to be stored (or striped) across multiple disks.

(Of course, several files may be accessed concurrently.)



# File Creation Rate on Snow and Blue

- Testing the creation time of 1000 files (or directories) created on NFS, JFS, and GPFS for Snow and Blue.
- Note: Blue runs GPFS 1.2, whereas Snow runs the newer GPFS 1.3




For file tests:

Machines: snow.llnl.gov
blue.llnl.gov

Code used: csh-script

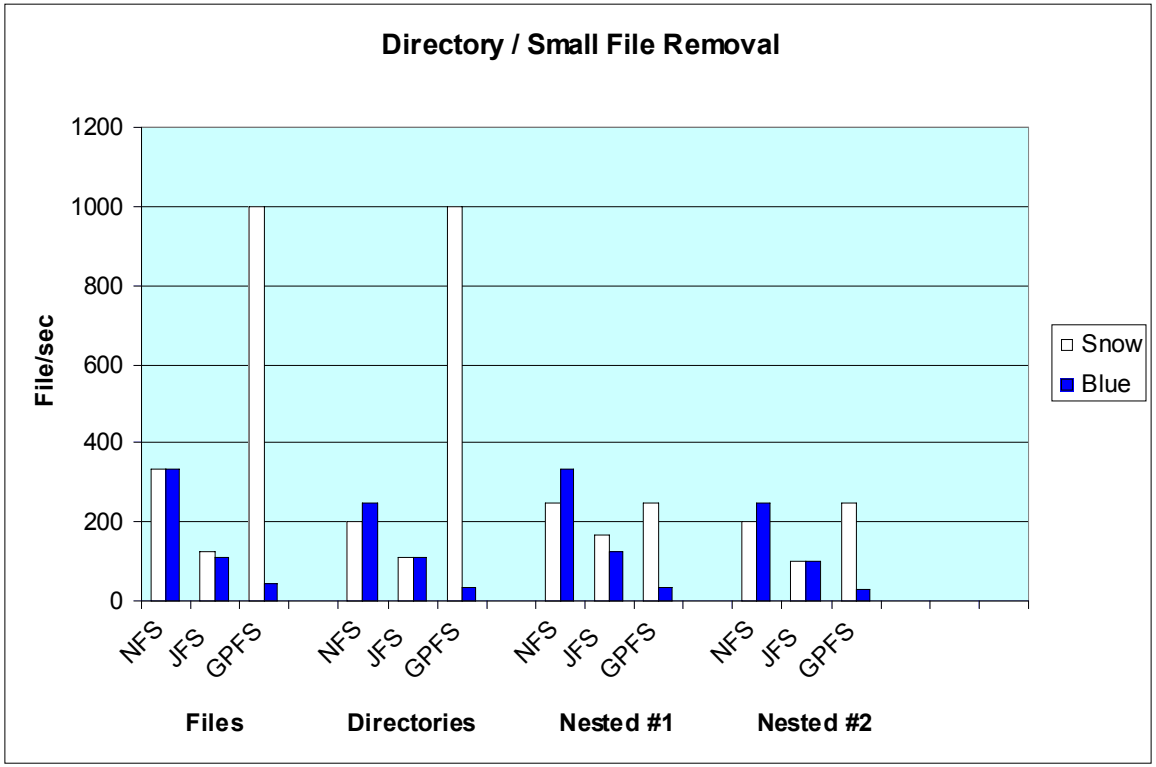
```
while ( $nfils < 1000 )
  echo "Sat Jan 01 00:00:00 PDT 2000" > file.$nfils
  @ nfils++
end
```

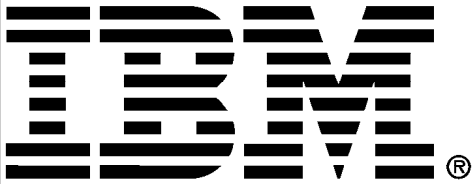
*Nested #1 -- Subdirectory containing one file and one subdirectory, recursively, for a 1000 file/directory total.

**Nested #2 -- Subdirectory containing nine files and one subdirectory, recursively, for a 1000 file/directory total.

# File Removal Rate on Snow and Blue

- GPFS 1.3 is significantly faster for removing files or directories than version 1.2





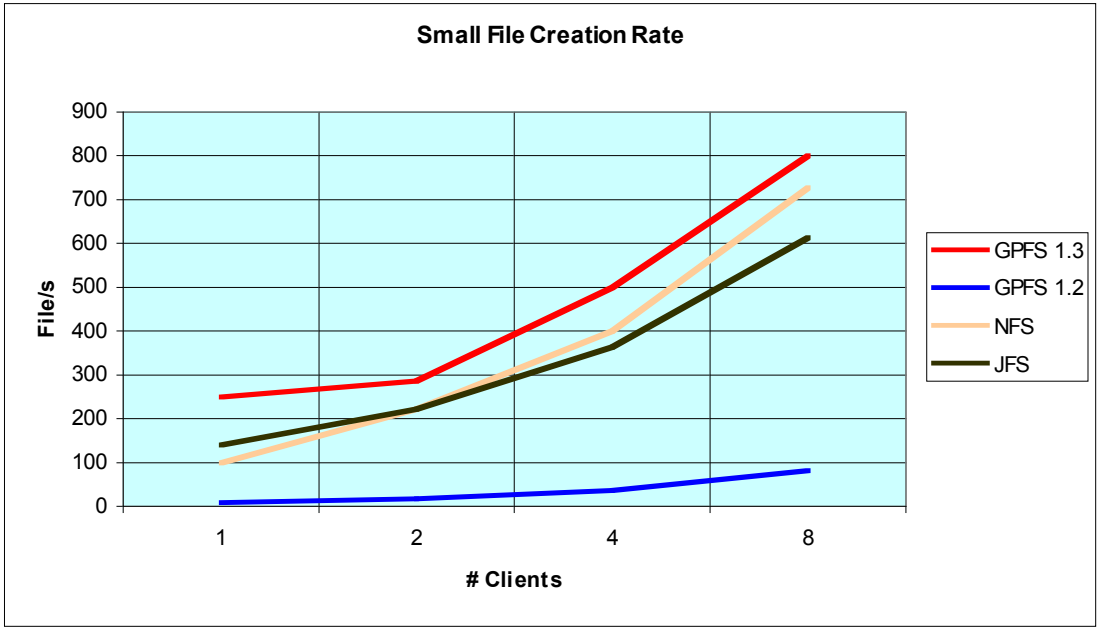
# File Creation/Removal Rate Summary for Snow and Blue


- The Network File System (NFS) and Journaled File System (JFS) are comparable on Snow and Blue (both using the same version of these file systems.) This shows that despite differences in hardware, the systems behave similarly. Therefore, the notable difference in GPFS performance is likely due to improvements between 1.2 and 1.3, not hardware considerations.

# Small File Creation Rate for GPFS 1.2, GPFS 1.3, NFS, JFS

**PARAMETERS:**

- 1000 29-byte files created on all file systems




Note:

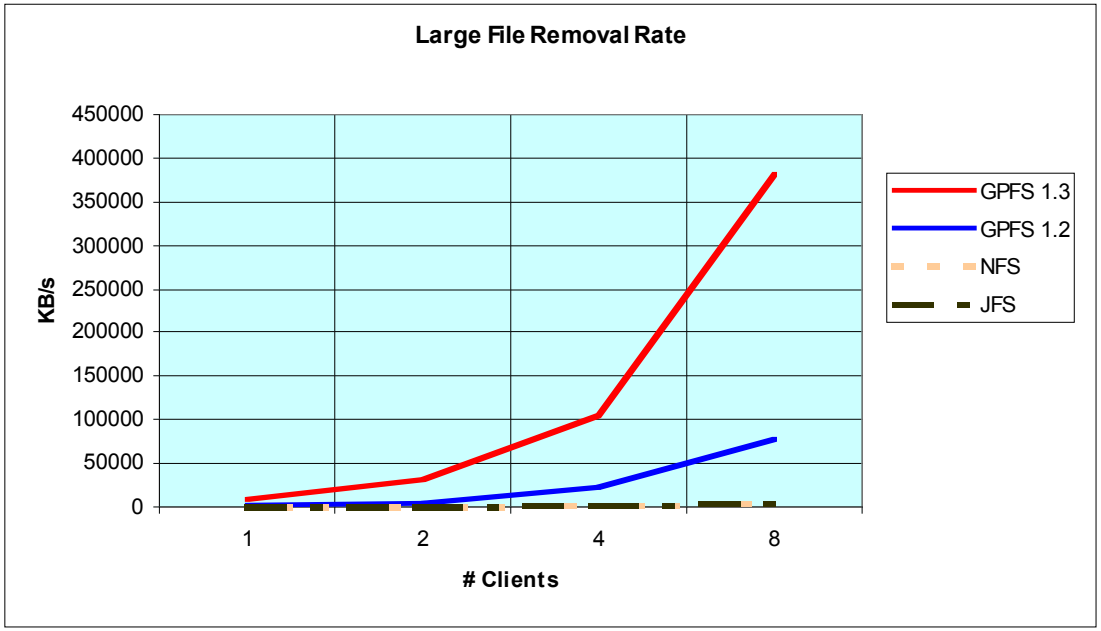
GPFS 1.2 tests run on blue.llnl.gov

GPFS 1.3, NFS, and JFS tests run on snow.llnl.gov

# Large File Removal Rate for GPFS 1.2, GPFS 1.3, NFS, JFS

**PARAMETERS:**

- GPFS 1.2: 128 1G files
- GPFS 1.3: 128 1G files
- NFS: 1000 1M files
- JFS: 1000 _M files




Note:

GPFS 1.2 tests run on blue.llnl.gov

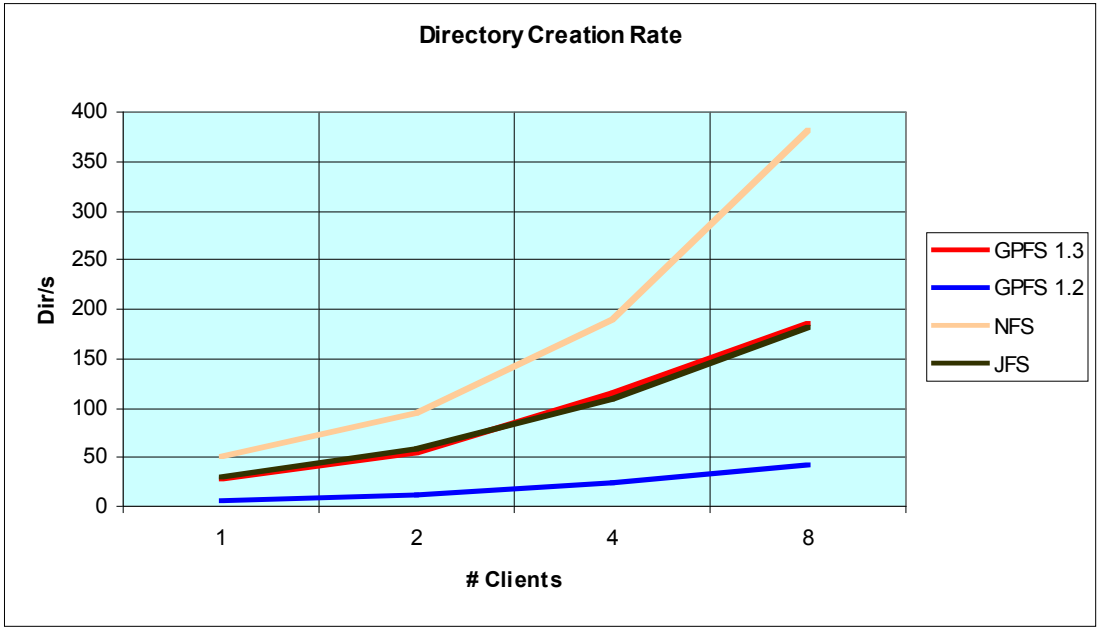
GPFS 1.3, NFS, and JFS tests run on snow.llnl.gov

# Directory Creation Rate for GPFS 1.2, GPFS 1.3, NFS, JFS

**PARAMETERS:**

- 1000 single-nested subdirectories created on all file systems

(*e.g.*, /1, /1/2, 1/2/3 would be three single-nested subdirectories, *i.e.*, a unary tree structure)




Note:

GPFS 1.2 tests run on blue.llnl.gov

GPFS 1.3, NFS, and JFS tests run on snow.llnl.gov

# Creation / Removal Rate for Snow for GPFS 1.2, GPFS 1.3, NFS, JFS Summary

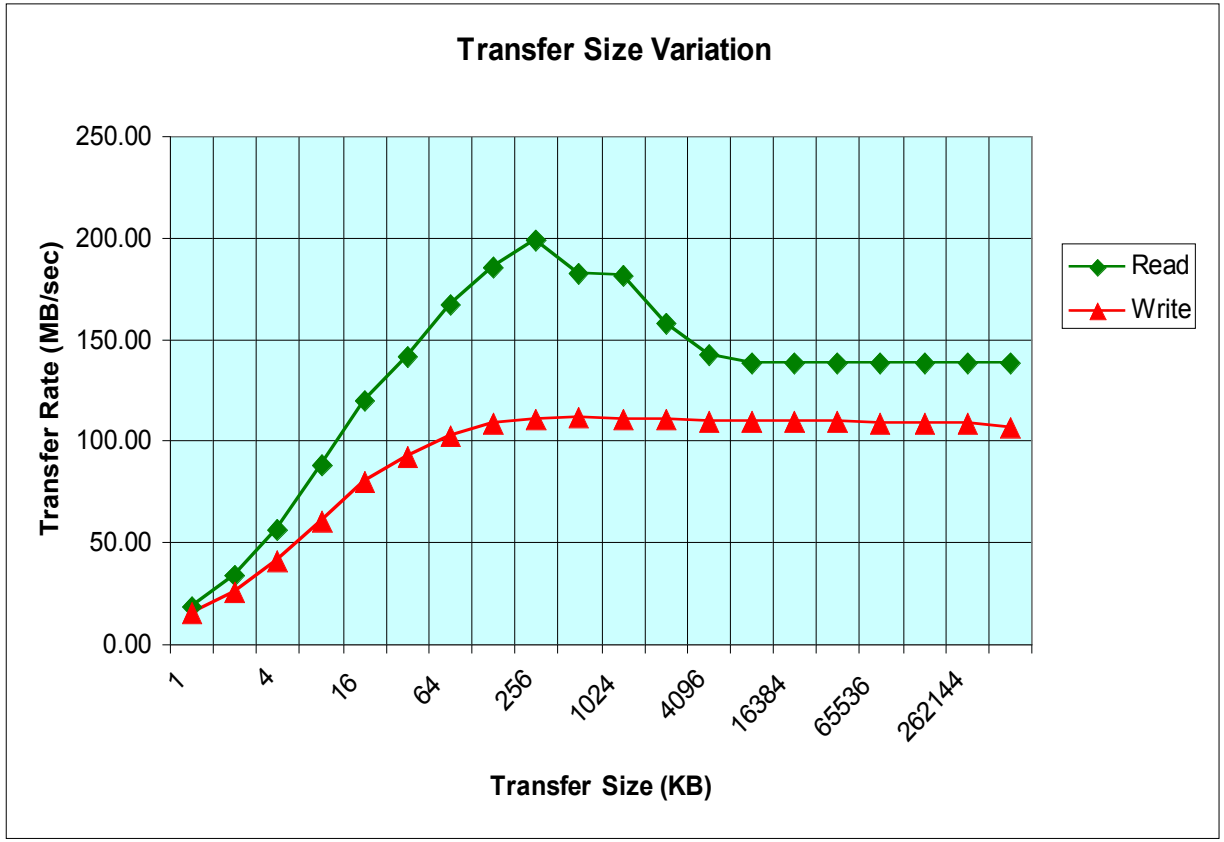

- The file creation and file removal rates are excellent on GPFS 1.3. Further, they appear scalable for increasing number of clients.

- With nested directories, the creation rate on GPFS 1.3 is below that of NFS.

# Sensitivity Curve I-A: Transfer Size Variation (Segmented)

**PARAMETERS:**

- Client = 1
- Node = 1
- FileSize = 512MB
- TransferSize = **1KB** to **512MB**




For all subsequent tests:

Code: ior_posix.c
Machine: snow.llnl.gov
PSSP: 3.2
GPFS: 1.3
Configuration: 2 * ( 3 * ( 3 * ( 4 + p )))
~250MB/sec Max Transfer Rate (14MB/sec per RAID set)

# RERUN: 11/2000
# Sensitivity Curve I-A:
# Transfer Size Variation (Segmented)

**PARAMETERS:**

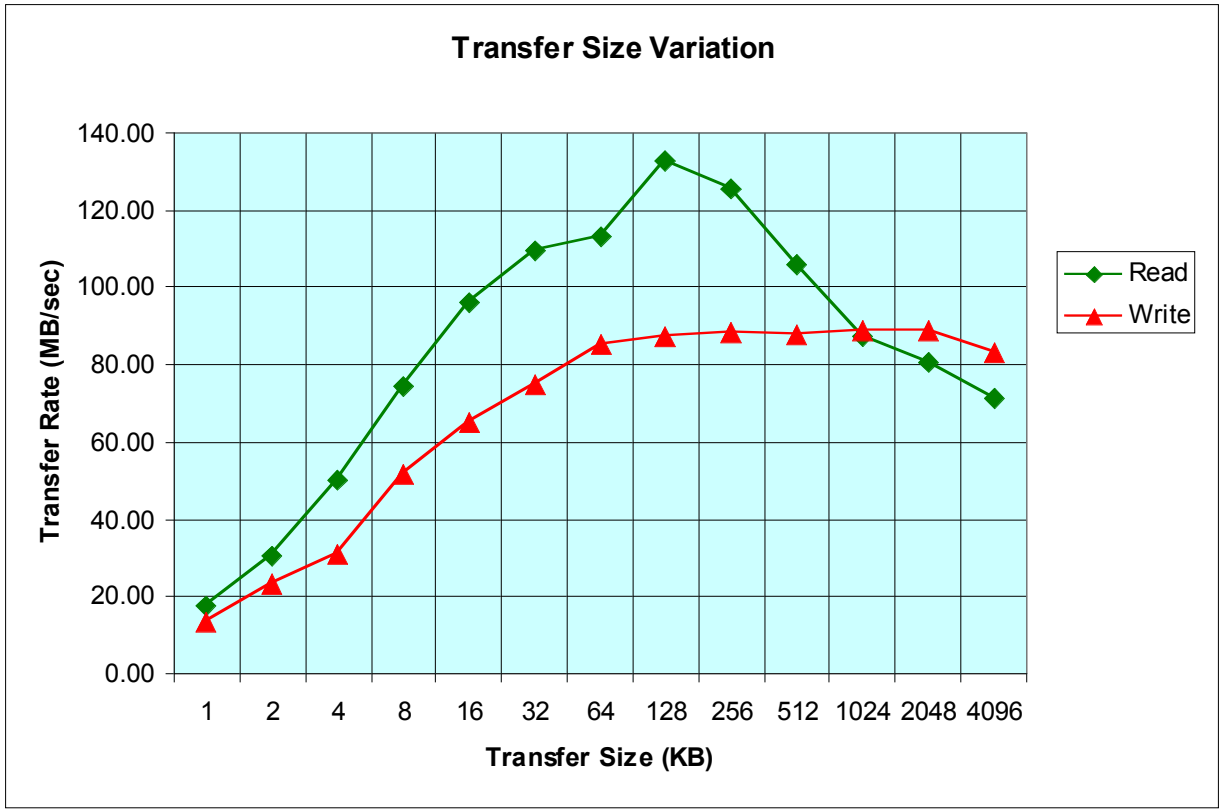
- Client = 1
- Node = 1
- FileSize = 512MB
- TransferSize =
  **1KB** to **512MB**

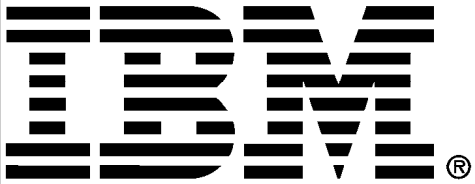
queue_depth = 2
previous run with
queue_depth = 40

# Sensitivity Curve I-B: Transfer Size Variation (Segmented)

**PARAMETERS:**

- Client = 1
- Node = 1
- FileSize = 5GB
- TransferSize = **1KB** to **4MB**





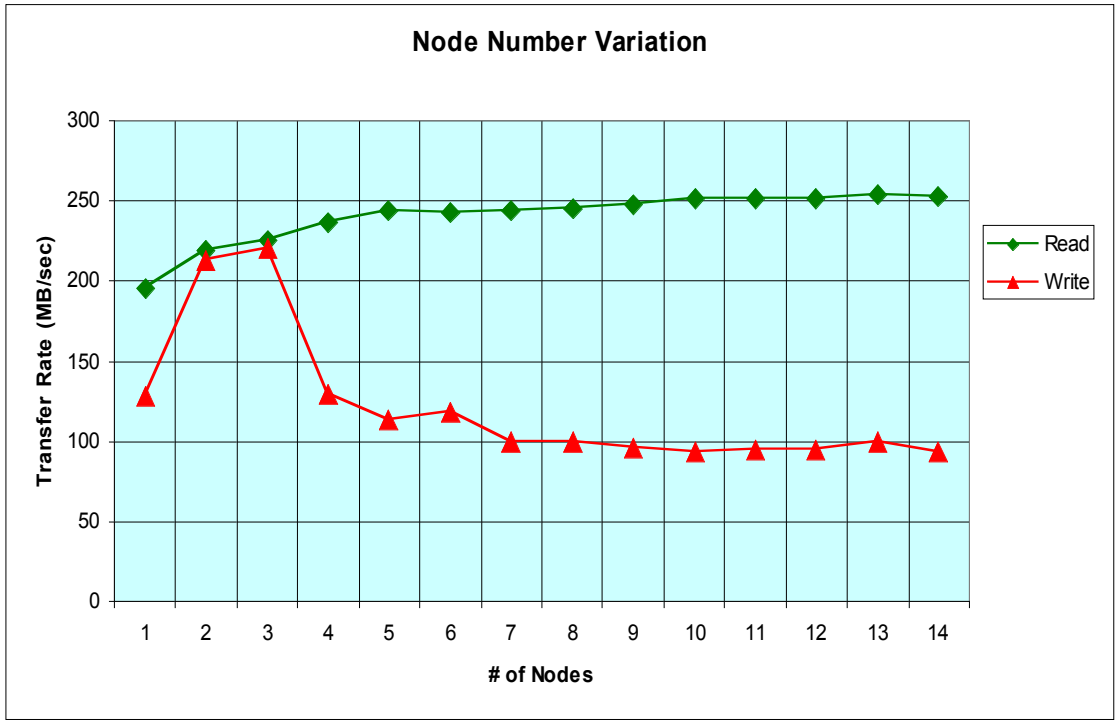
# Transfer Size Variation Summary


- For reads, the read-ahead algorithm GPFS uses becomes inefficient for larger subblocks, thus causing the bell-curve for reads. The bell-shaped curve is caused by the filling of the L2 cache which bumps out the instruction stream with larger transfer sizes.
- Writes can be expected to be slower due to the additional overhead associated with allocation (buffer must be allocated before write request.) But nonetheless for writes, the ceiling of 140MB/sec by one thread (CPU) per Node supposedly causes the poor write performance. Later, we can show that this appears to be the case, but that another bottleneck is creeping up elsewhere.

# Sensitivity Curve II-A: Node Number Variation (Segmented)

**PARAMETERS:**

- Client / Node = 1
- TransferSize = 256KB
- Block = 512MB
- FileSize = Client * 512MB
- Nodes = **1** to **14**

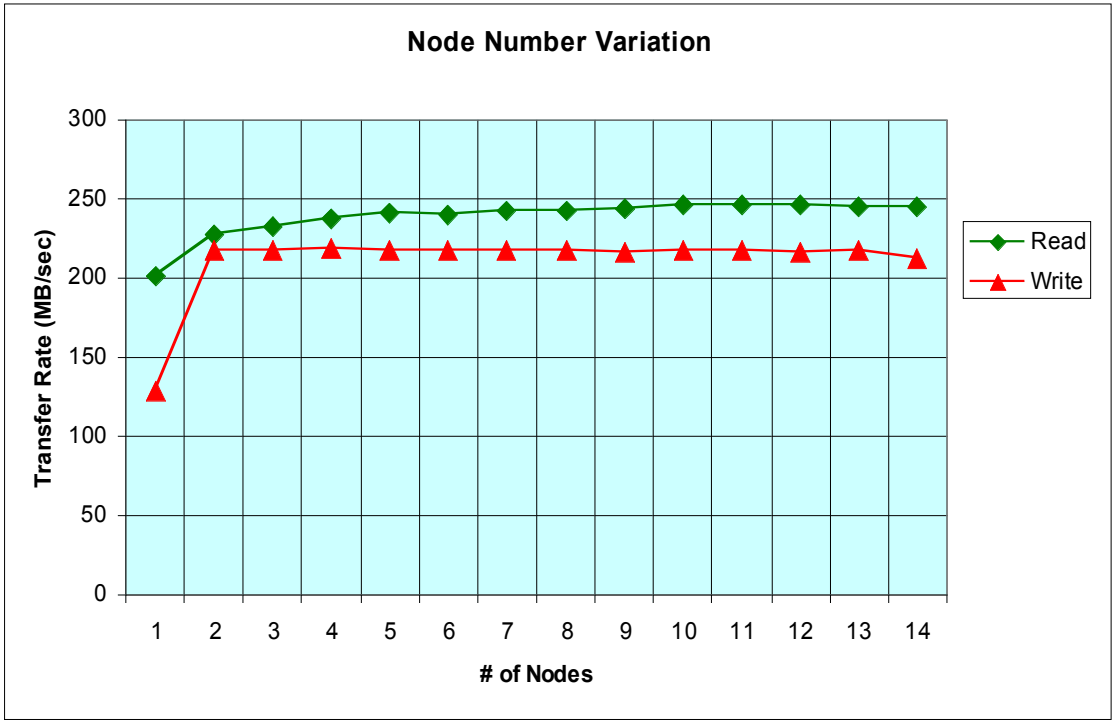


# RERUN: 11/2000
# Sensitivity Curve II-A:
# Node Number Variation (Segmented)

**PARAMETERS:**

- Client / Node = 1
- TransferSize = 256KB
- Block = 512MB
- FileSize = Client * 512MB
- Nodes = **1** to **14**

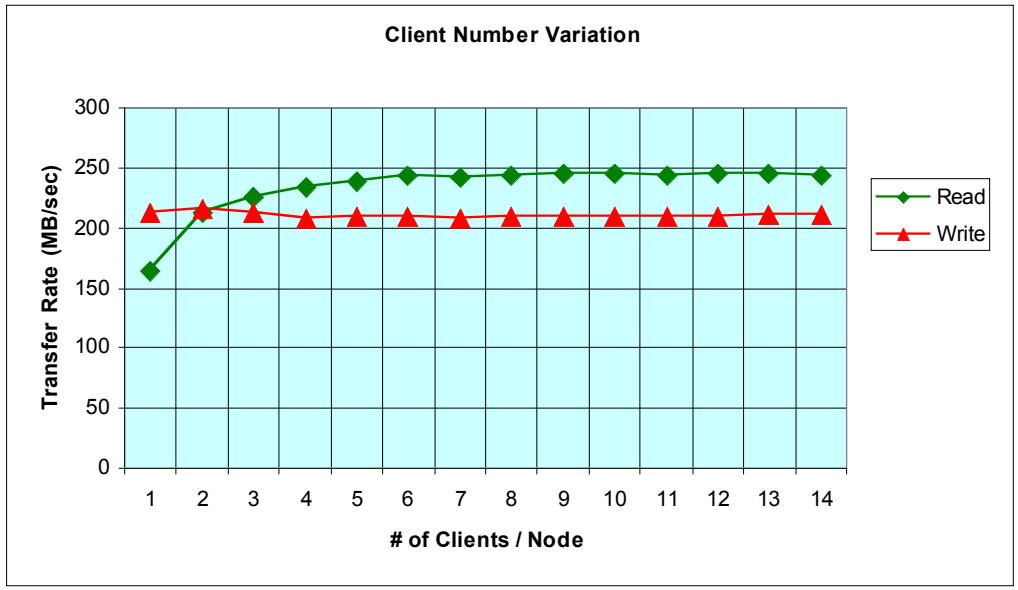
queue_depth = 2
previous run with
queue_depth = 40



# RERUN: 11/2000
# Sensitivity Curve II-A2:
# Node Number Variation (Segmented)

**PARAMETERS:**

- Client / Node = 8
- TransferSize = 256KB
- Block = 512MB
- FileSize = Client * 512MB
- Node = **1** to **14**



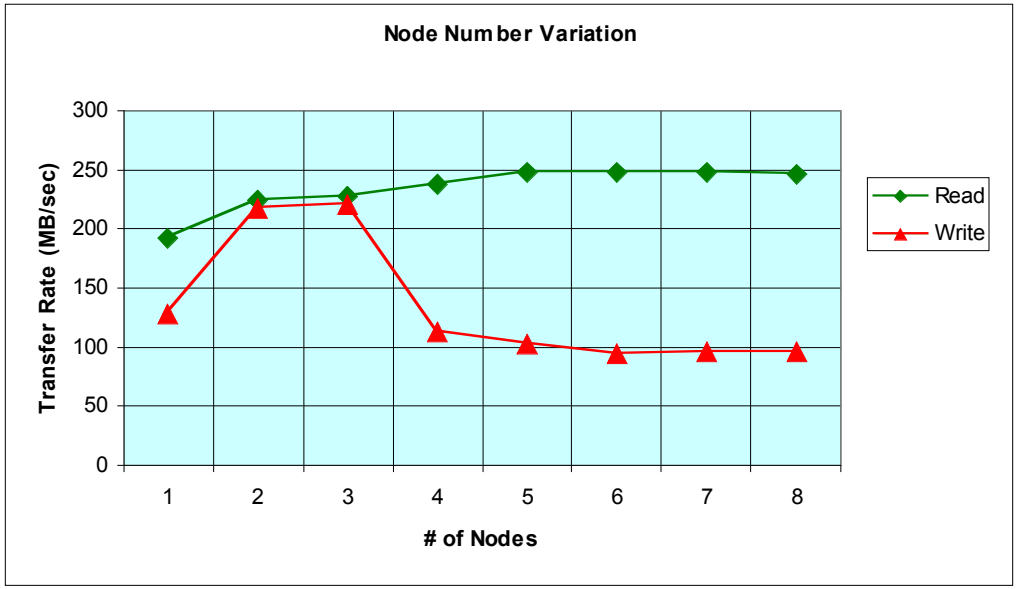

queue_depth = 2
previous run with
queue_depth = 40

**[NOTE: No Summer run of this test for comparision]**

# Sensitivity Curve II-B: Node Number Variation (Segmented)

**PARAMETERS:**

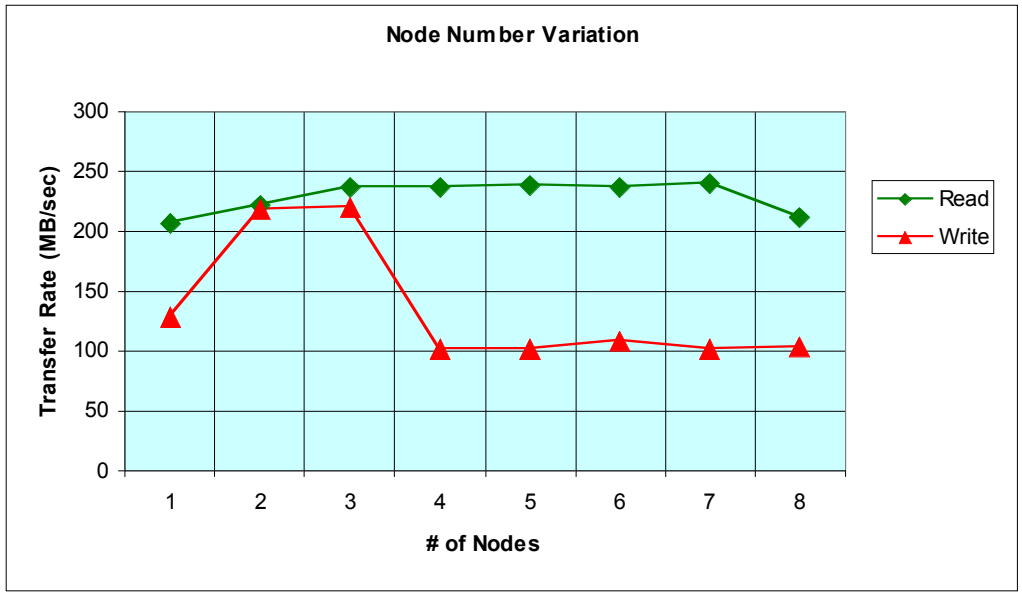
- Client / Node = 1
- FileSize = 8400MB
- TransferSize = 256KB
- Block = FileSize / Clients
- Nodes = **1** to **8**



# Sensitivity Curve II-C: Node Number Variation (Segmented)

**PARAMETERS:**

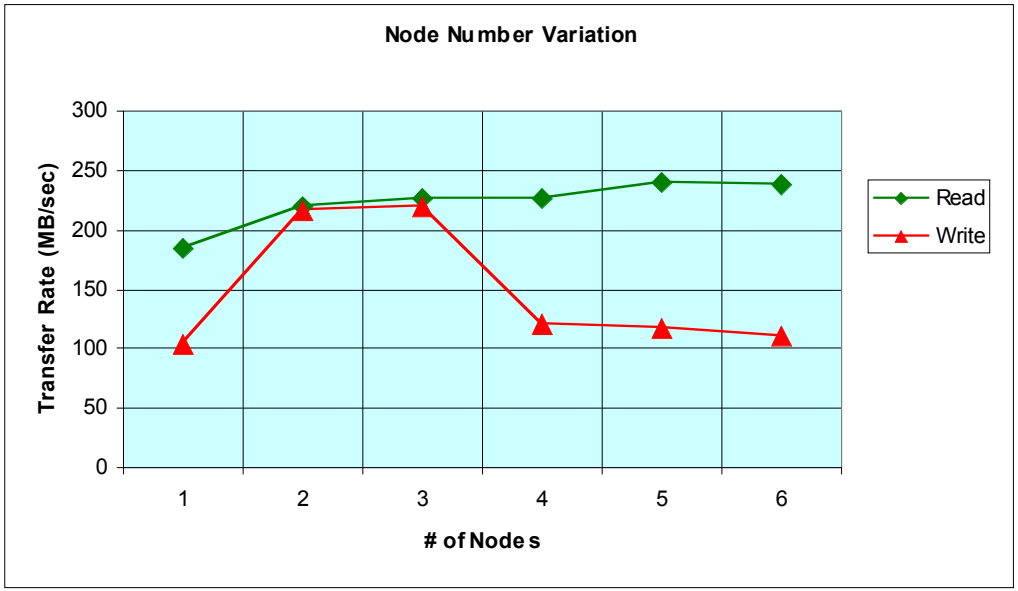
- Client / Node = 1
- FileSize = 2100MB
- TransferSize = 256KB
- Block = FileSize / Clients
- Nodes = **1** to **8**

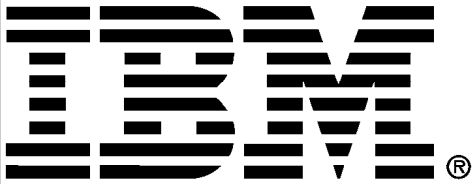


# Sensitivity Curve II-D: Node Number Variation (Segmented)

**PARAMETERS:**

- Client / Node = 1
- FileSize = 1470MB
- TransferSize = 256KB
- Block = FileSize / Clients
- Nodes = **1** to **6**





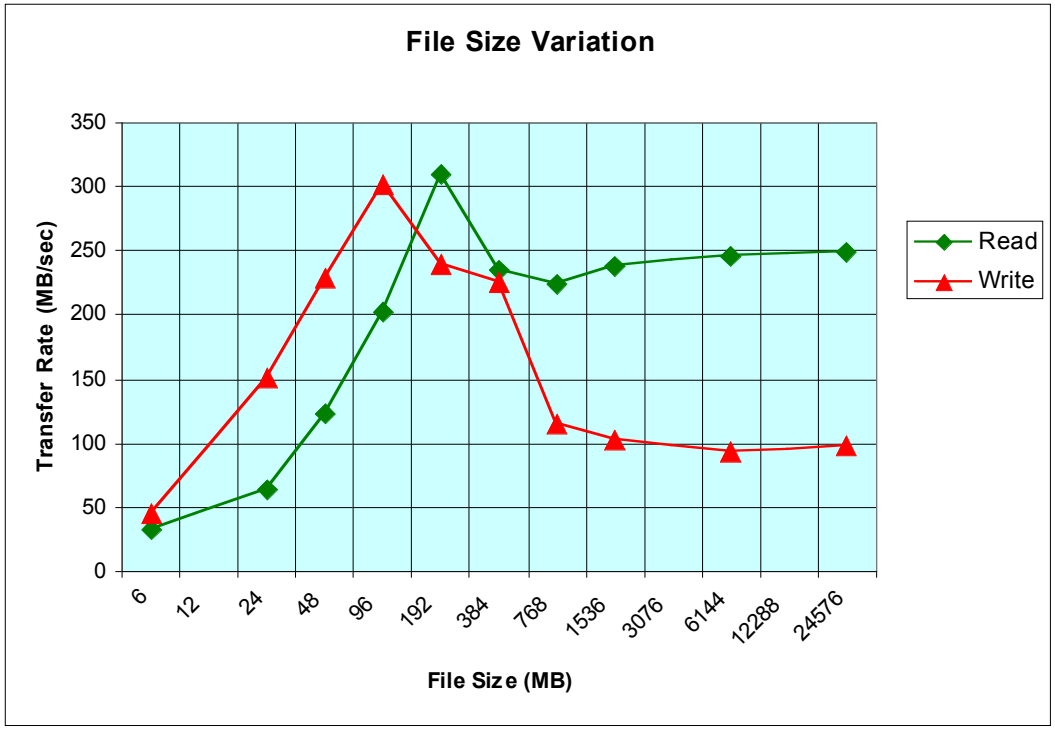
# Node Number Variation Summary


- It appears from observation of the activity of the I/O nodes that a “read-modify-write” is being performed when using more than 3 clients during this test. As the RAID stripe is 256KB, but the LVM transfer size is 128KB, there is a failure to coalesce the transfer packet to 256KB before writing to disk. Consequently, a read-modify-write is performed, slowing down the write.

# Sensitivity Curve III: File Size Variation (Segmented)

**PARAMETERS:**

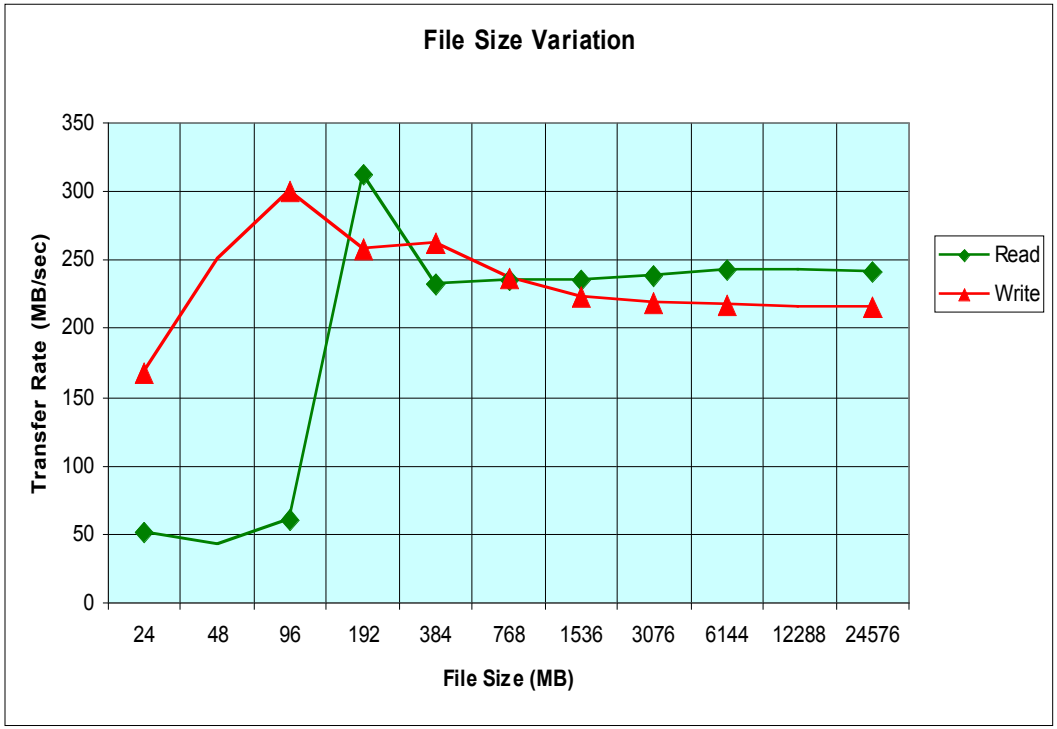
- Nodes = 6
- Client / Node = 1
- TransferSize = 256KB
- Block = FileSize / 6
- FileSize =**6 MB** to **24GB**

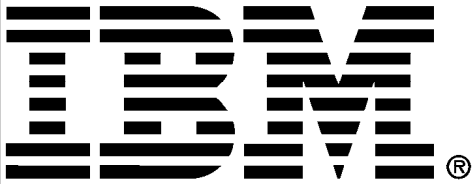


# RERUN: 11/2000
# Sensitivity Curve III:
# File Size Variation (Segmented)

**PARAMETERS:**

- Nodes = 6
- Client / Node = 1
- TransferSize = 256KB
- Block = FileSize / 6
- FileSize =**6 MB** to **24GB**





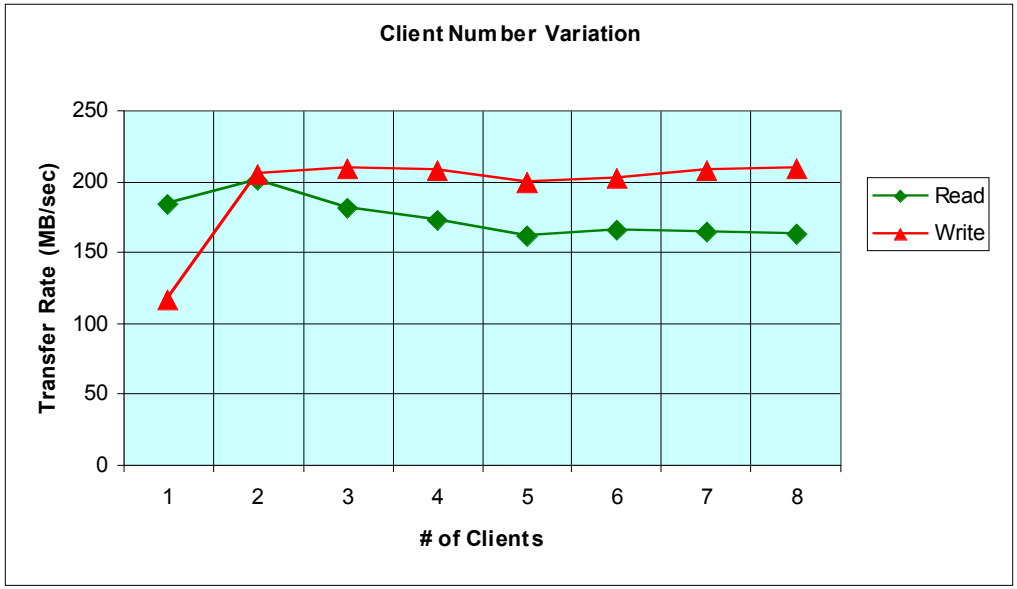
# File Size Variation Summary


- Apparently clever use of caching allows for a better read/write rate. But, after a certain blocksize as the file gets larger, the read levels off to around 250MB/sec, but the write rate plummets to 100MB/sec. These are the same bottlenecks we've been seeing.
- However, note that writing 32/64MB per node can help boost GPFS through use of caching. As the Page Pool is set to 100MB, we do not see the effects of caching beyond this point.

# Sensitivity Curve IV-A: Client / Node Variation (Segmented)

**PARAMETERS:**

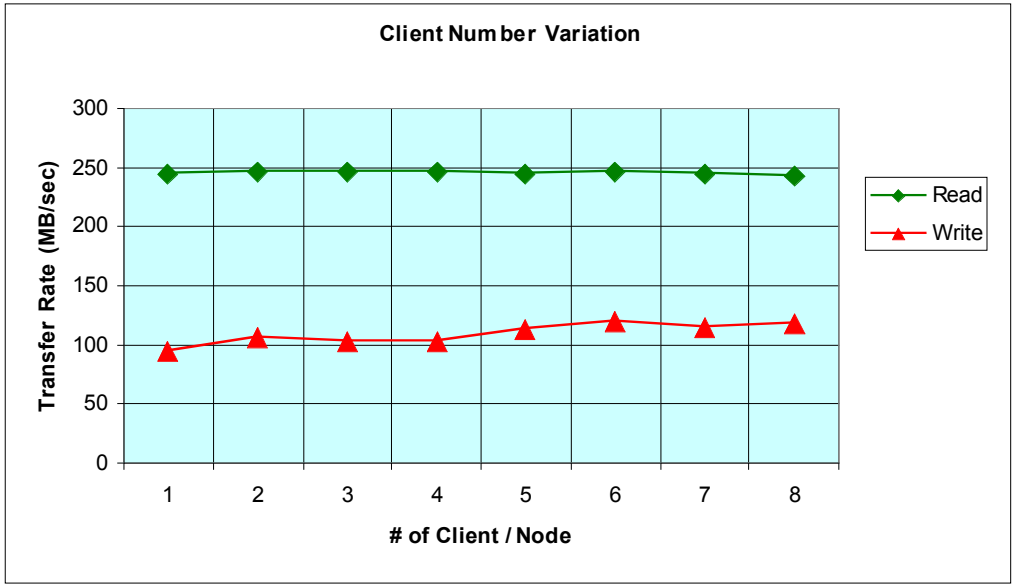
- Nodes = 1
- FileSize = 512MB * Clients
- TransferSize = 256KB
- Block = 512MB
- Clients = **1** to **8**



# Sensitivity Curve IV-B: Client / Node Variation (Segmented)

**PARAMETERS:**

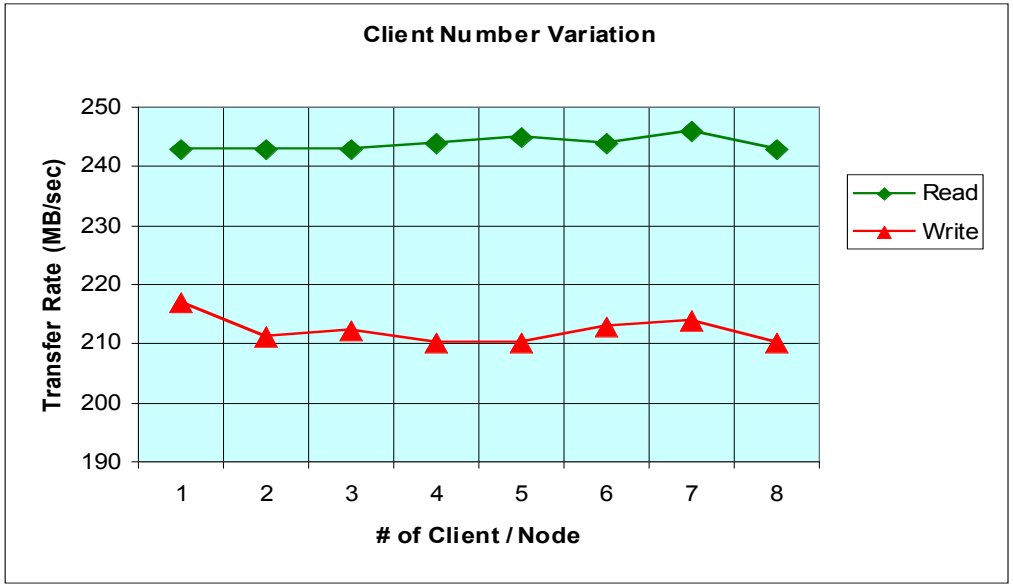
- Nodes = 8
- TransferSize = 256KB
- Block = 512MB
- FileSize = Client * 512MB
- Client / Node = **1** to **8**



# RERUN: 11/2000
# Sensitivity Curve IV-B:
# Client / Node Variation (Segmented)

**PARAMETERS:**

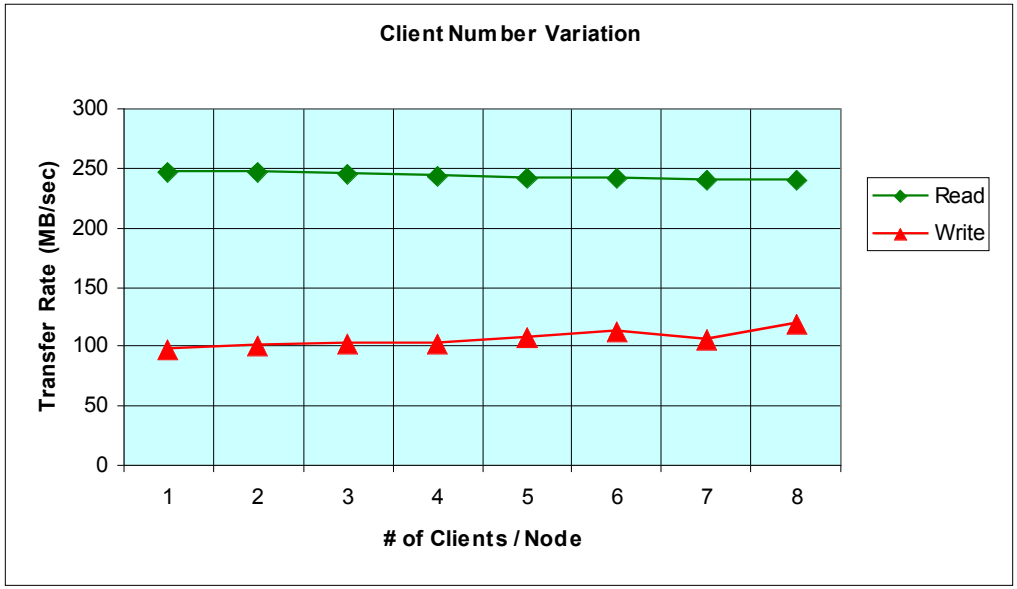
- Nodes = 8
- TransferSize = 256KB
- Block = 512MB
- FileSize = Client * 512MB
- Client / Node = **1** to **8**

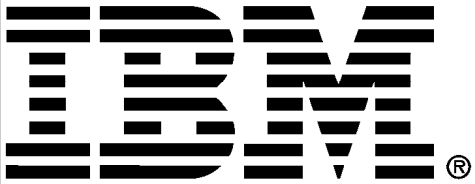


# Sensitivity Curve IV-C: Client / Node Variation (Segmented)

**PARAMETERS:**

- Nodes = 8
- FileSize = 8400MB
- TransferSize = 256KB
- Block = FileSize / (total) Clients
- Client / Node = **1** to **8**





# Client / Node Variation Summary

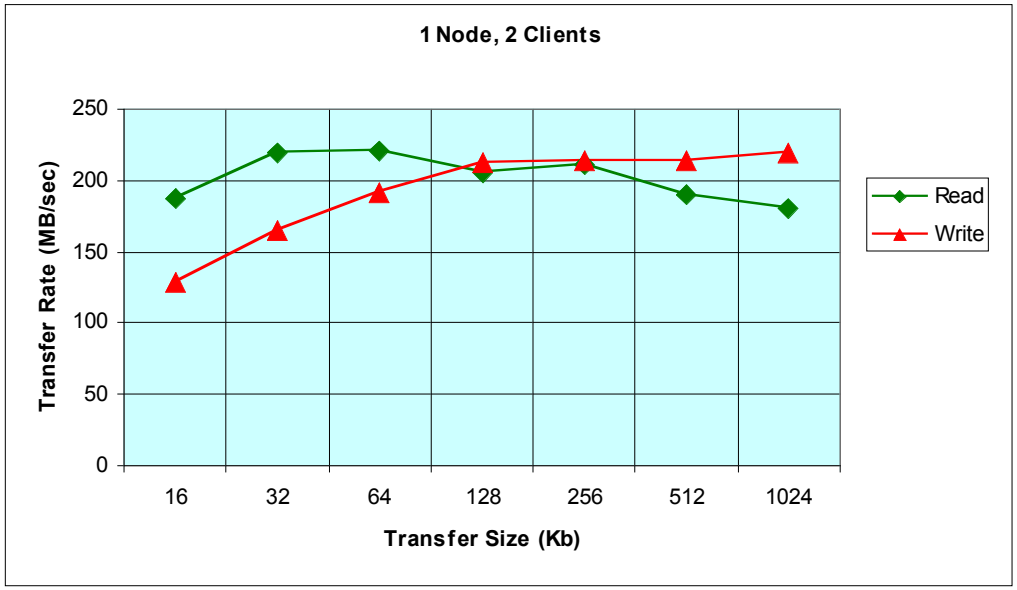

- With one node, the write rate max of 140 MB/sec increases after one client.
- This does not hold true, however, with more nodes. Instead, the write rate stays low. Perhaps a different bottleneck is in effect here.

# Sensitivity Curve V-A: Multiple Clients with Transfer Size Variation (Segmented)

**PARAMETERS:**

- Clients = 2
- Nodes = 1
- FileSize = 1024MB
- Block = 512MB
- TransferSize =

  **16KB – 1024KB**

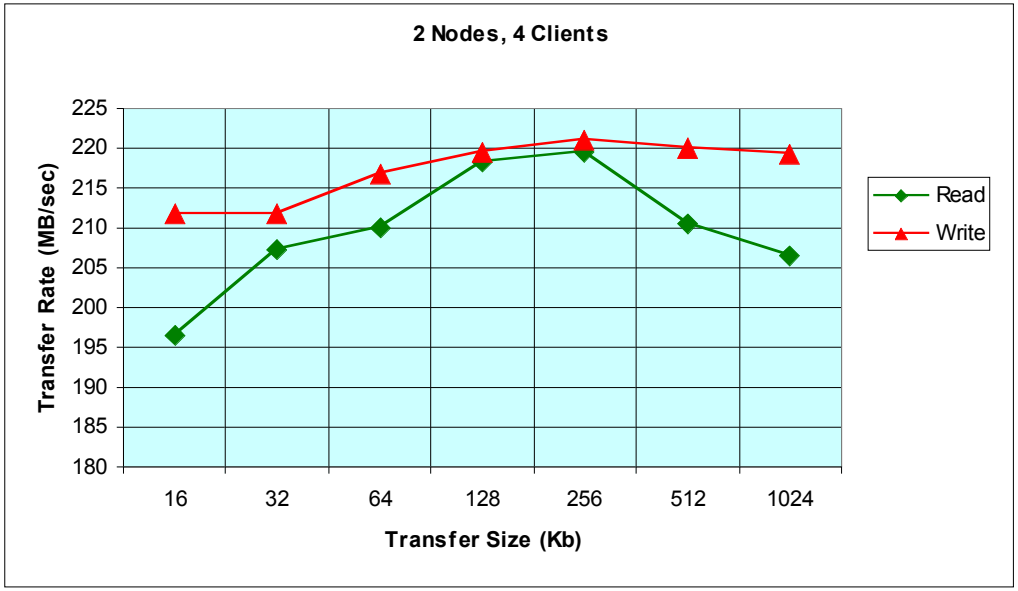


# Sensitivity Curve V-B: Multiple Clients with Transfer Size Variation (Segmented)

**PARAMETERS:**

- Clients = 4
- Nodes = 2
- FileSize = 2048MB
- Block = 512MB
- TransferSize =

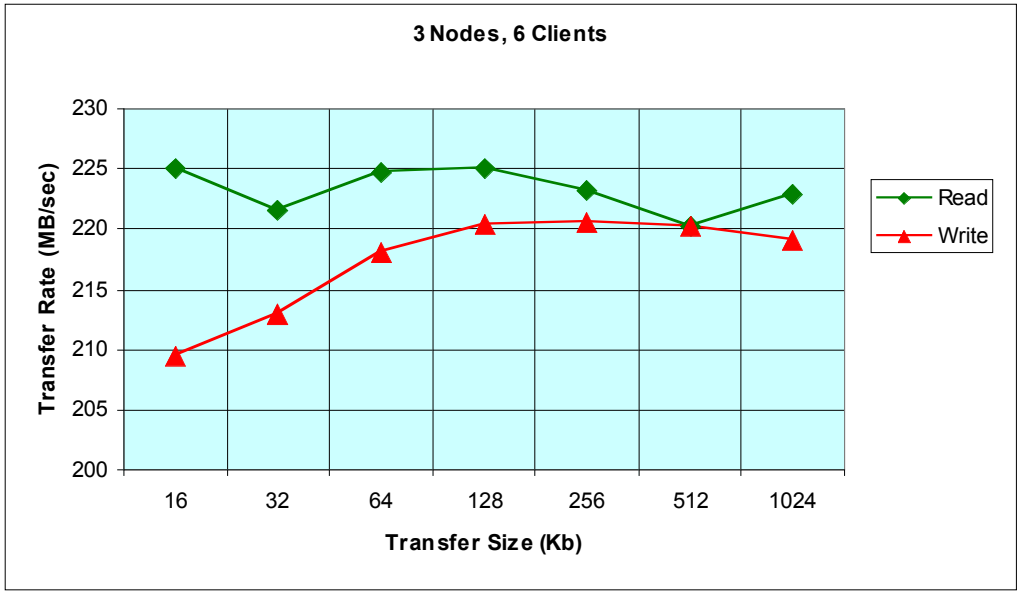
  **16KB – 1024KB**



# Sensitivity Curve V-C: Multiple Clients with Transfer Size Variation (Segmented)

**PARAMETERS:**

- Clients = 6
- Nodes = 3
- FileSize = 3072MB
- Block = 512MB
- TransferSize = **16KB – 1024KB**

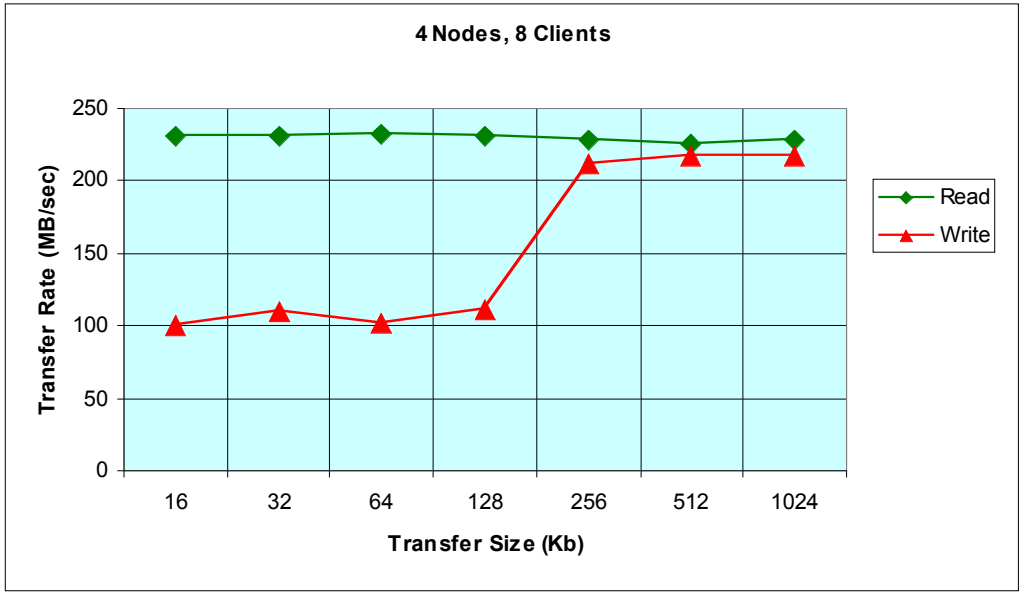


# Sensitivity Curve V-D: Multiple Clients with Transfer Size Variation (Segmented)

**PARAMETERS:**

- Clients = 8
- Nodes = 4
- FileSize = 4096MB
- Block = 512MB
- TransferSize =

  **16KB – 1024KB**

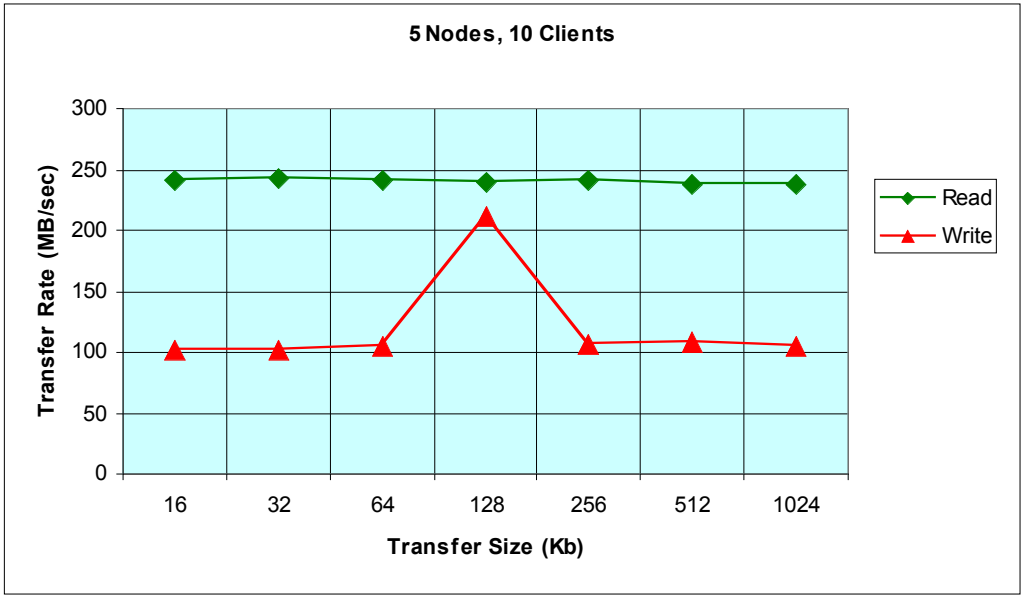


# Sensitivity Curve V-E: Multiple Clients with Transfer Size Variation (Segmented)

**PARAMETERS:**

- Clients = 10
- Nodes = 5
- FileSize = 5120MB
- Block = 512MB
- TransferSize =

  **16KB – 1024KB**

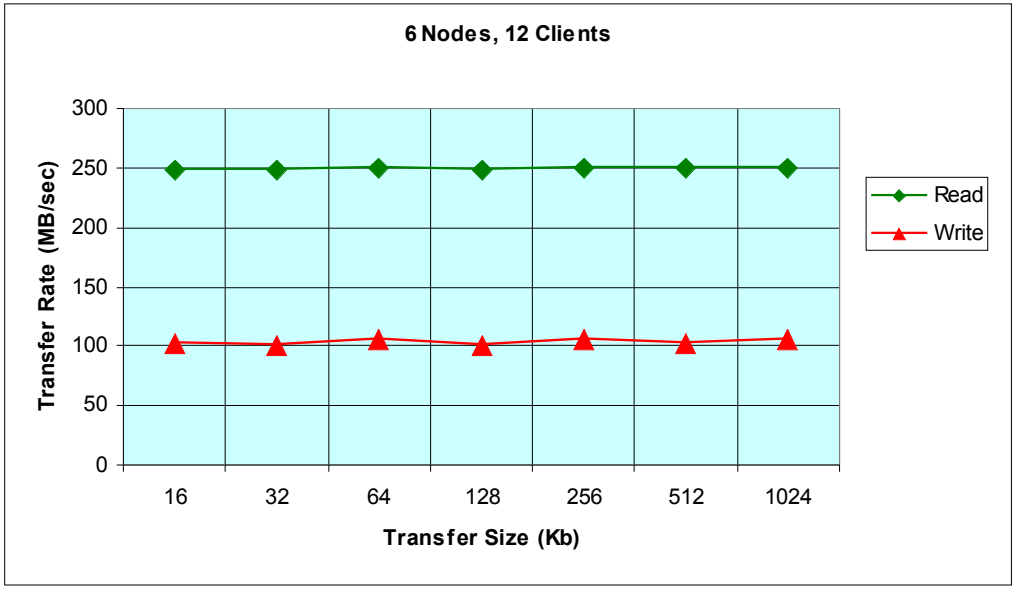


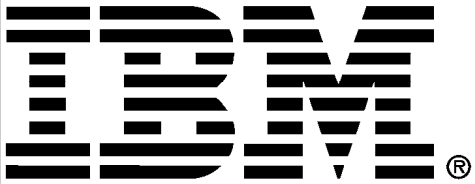
# Sensitivity Curve V-F: Multiple Clients with Transfer Size Variation (Segmented)

**PARAMETERS:**

- Clients = 12
- Nodes = 6
- FileSize = 6144MB
- Block = 512MB
- TransferSize =

  **16KB – 1024KB**





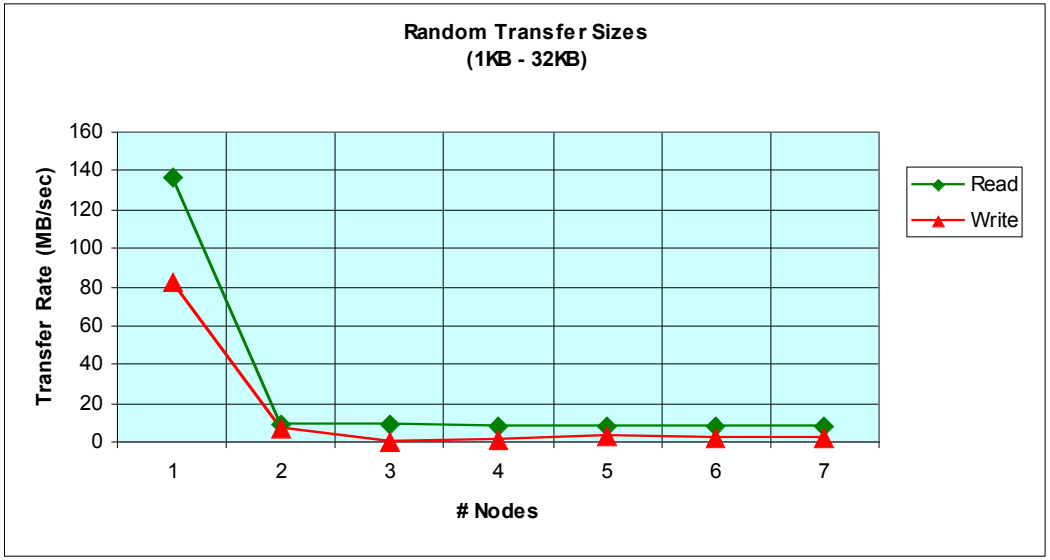
# Multiple Clients with Transfer Size Variation Summary

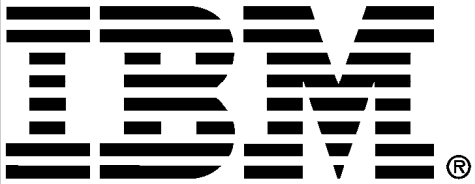

- To determine the effect of the 140MB/sec per CPU bottleneck, the benchmark is run with increasing the number of nodes. With four or few nodes, the write rate is good for any size transfer. After that, however, something slows the write rate. It seems the ‘camel hump’ from Curve II is consistent.

# Sensitivity Curve VI-A: Random Transfer Size (Strided)

**PARAMETERS:**

- Client / Node = 1
- FileSize = Clients * 512MB
- TransferSize = **1KB** to **32KB**
- Nodes = **1** to **7**





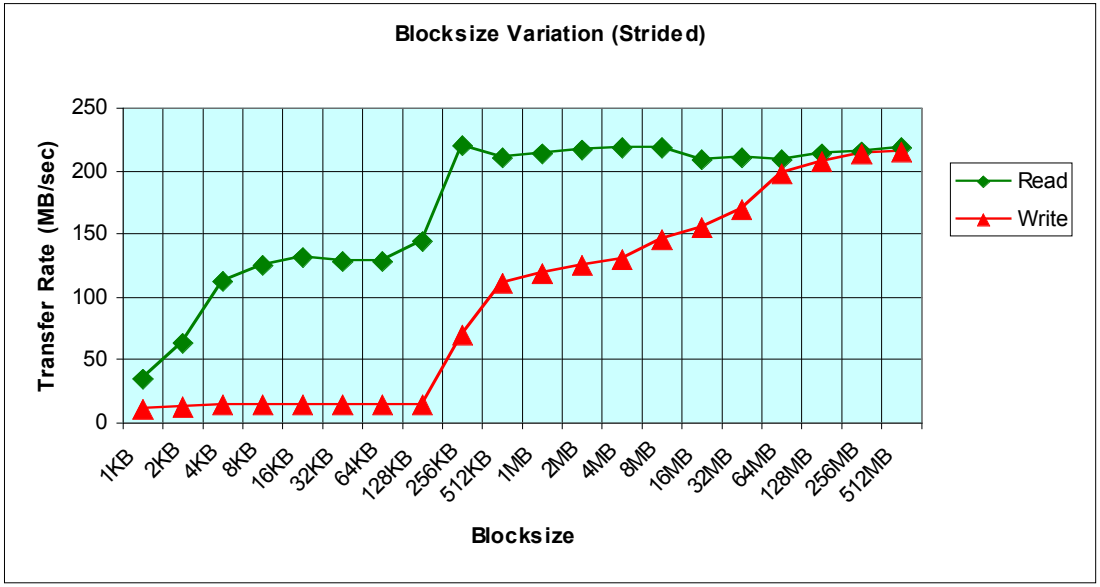
# Random Transfer Size Summary


- As suspected, IOR does not address the problems with random transfer sizes very well. Hopefully MPI-IO will improve these rates.

# Sensitivity Curve VII-A: Blocksize Variation (Strided)

**PARAMETERS:**

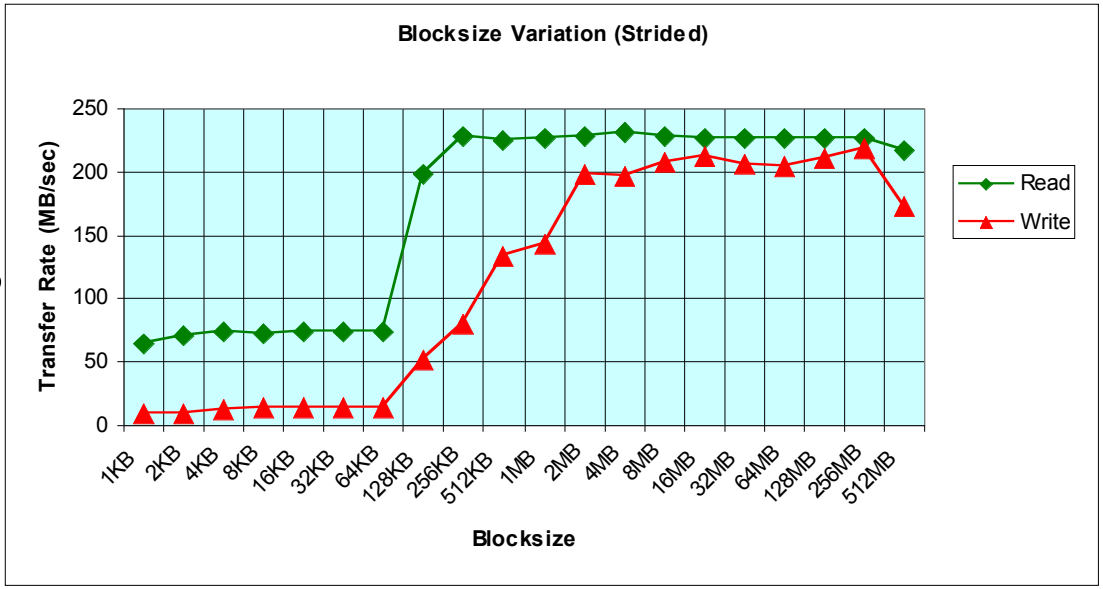
- Nodes = 2
- Client / Node = 1
- FileSize = Clients * 512MB
- Block = **1KB** to **512MB**
- (Note: Strided pattern)



# Sensitivity Curve VII-B: Blocksize Variation (Strided)

**PARAMETERS:**

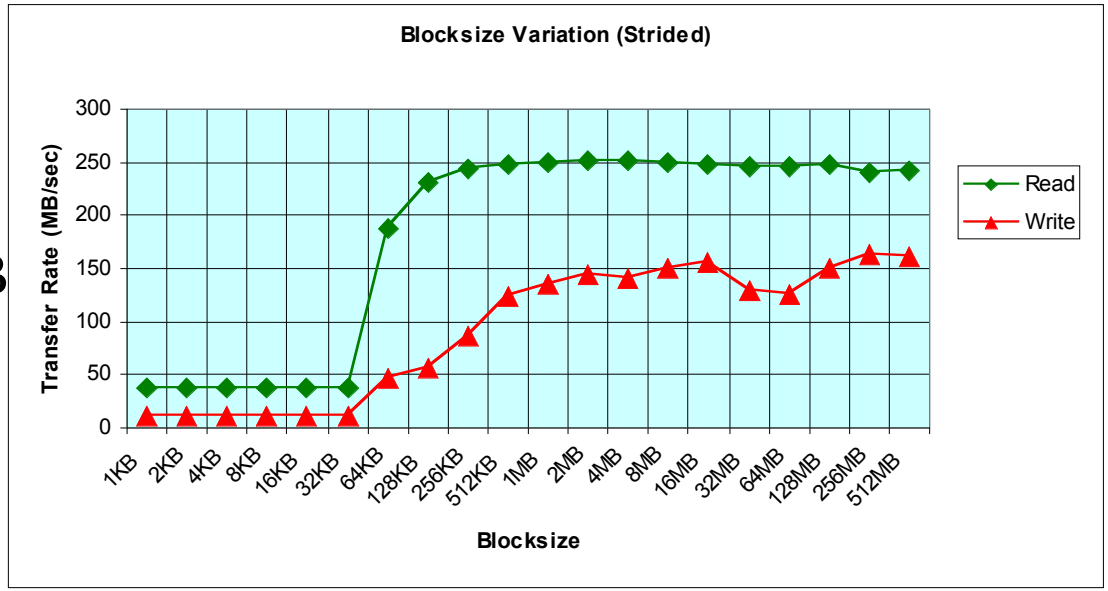
- Nodes = 4
- Client / Node = 1
- FileSize = Clients * 512MB
- Block = **1KB** to **512MB**



# Sensitivity Curve VII-C: Blocksize Variation (Strided)

**PARAMETERS:**

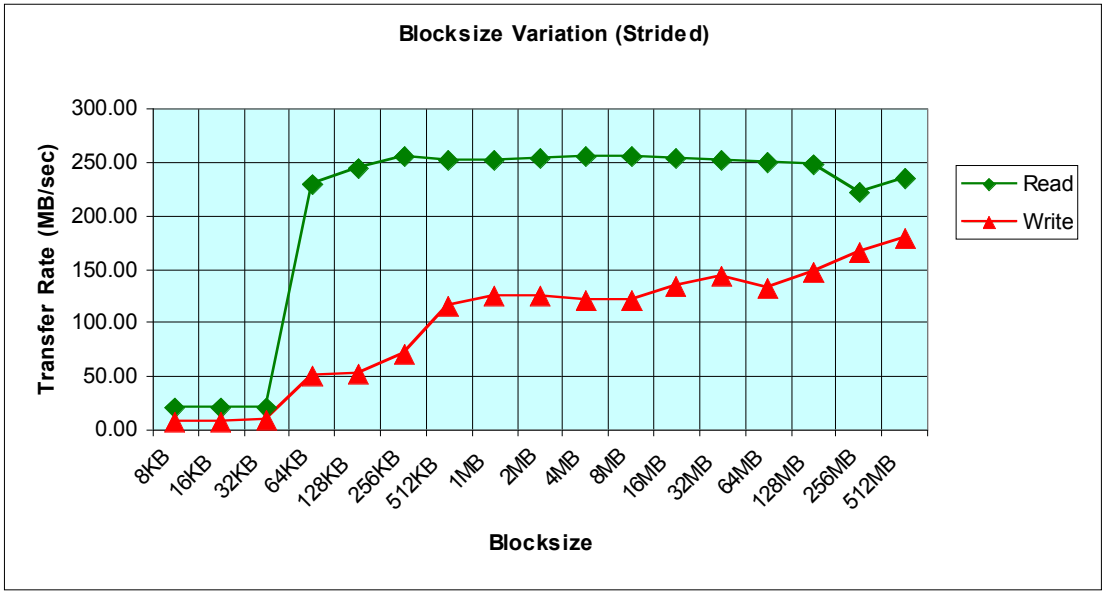
- Nodes = 8
- Client / Node = 1
- FileSize = Clients * 512MB
- Block = **1KB** to **512MB**



# Sensitivity Curve VII-D: Blocksize Variation (Strided)

**PARAMETERS:**

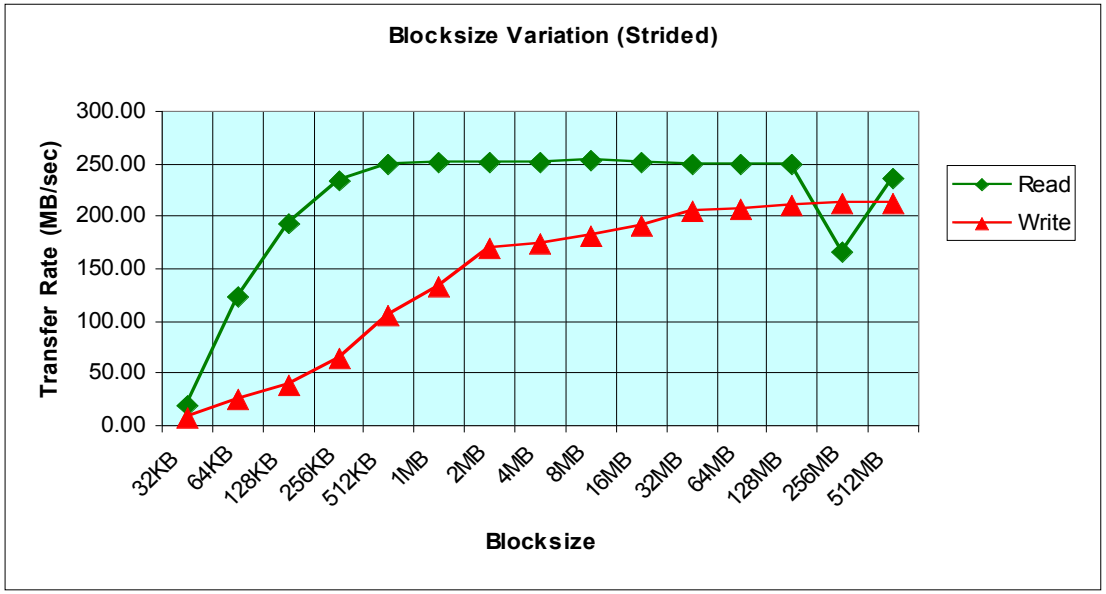
- Nodes = 14
- Client / Node = 1
- FileSize = Clients * 512MB
- Block = **8KB** to **512MB**

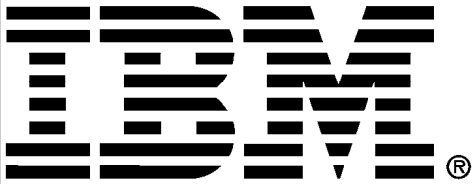


# RERUN: 11/2000
# Sensitivity Curve VII-D:
# Blocksize Variation (Strided)

**PARAMETERS:**

- Nodes = 14
- Client / Node = 1
- FileSize = Clients * 512MB
- Block = **32KB** to **512MB**





# Blocksize Variation Summary


- For strided reads above ~256KB block size (and corresponding transfer size), the performance is excellent for any number of nodes. In fact, we've achieved the theoretical bottleneck of ~250MB/sec.
- Writes, on the other hand, tend to be better with fewer nodes and tend to improve with larger block sizes. Initially (below 256KB) this is the read-modify-write problem. Beyond it, however, probably the large number of client are showing the coalescing problem again.

# Concluding Summary


**META-DATA (File Create/Delete) Summary:**

Changes in GPFS 1.3 have improved meta data performance greatly. Small file creation and large file removal are above that of NFS, JFS, and GPFS 1.2. For directory creation, GPFS 1.3 is strong, but still half that of NFS.

**IOR Benchmark Summary:**

I. An optimal transfer buffer size seems be ~256KB for reads and writes.

II. Read performance is excellent with a ceiling of ~250MB/sec. Increasing the number of nodes only marginally improves this read performance.

For writes, there is a coalescing bottleneck.

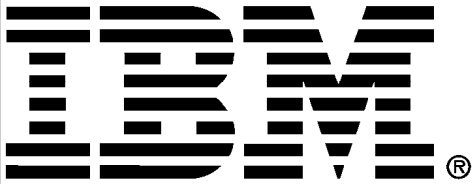
**IOR Benchmark Summary (cont.):**

III. The pagepool can improve read/write rates beyond the theoretical ceiling.

IV/V. Increasing the number of nodes using more than one client per node improves write rate, but again the > 4 node bottleneck is present.

VI. Reads/writes for random transfer size is less than desirable. MPI-IO will better address this.

VII. Strided reads greater than 256KB blocks are excellent across the board. Writes tend to ramp us slowly with improvements due to increasing blocksize. Again, writes are better with fewer nodes.



# Acknowledgements



- This Science & Technology Education Program summer project was completed under the guidance and supervision of Kim Yates and Terry Jones.

- Assistance and information from Dave Fox, Robin Goldstone, Mark Grondona, and Dan McNabb (of IBM) helped to clarify the results and offer general understanding of snow.llnl.gov and the GPFS system.

- This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.